Esercizio 11: divisore restoring

L’obiettivo di questo esercizio è quello di progettare, tramite un approccio strutturale, un divisore che ci permetta di effettuare una divisione tra un dividendo espresso su M bit ed un divisore, espresso su N bit. Il risultato sarà composto da un resto, codificato su al più N bit, ed un quoziente, espresso al più su M-N+1 bit (supponendo che il primo degli m/n bit non sia nullo).

Analizziamo ora il procedimento di divisione, per comprendere quali sono i componenti da utilizzare:

* Passo iniziale: si confrontano gli n bit più significativi del dividendo (dividendo parziale) con gli n bit del divisore: se il dividendo parziale è contenuto nel divisore, allora il quoziente parziale (qi) sarà pari ad 1, 0 altrimenti (Il risultato delle operazioni di divisione parziale può essere al massimo 1, quindi in binario si ha un grande vantaggio).

Si effettua quindi la sottrazione tra il dividendo parziale ed il prodotto qiV (dove V rappresenta il divisore), ottenendo così il primo resto parziale.

* Generico passo i: si pone il dividendo parziale Di pari al resto parziale Ri (calcolato all’iterazione i-1) e si confronta con il divisore. A questo punto i passaggi successivi sono analoghi a quelli riportati al passo 1, con l’unica differenza che il resto parziale andrà riportato shiftato a destra di i posizioni.

La formula generale per calcolare i resti parziali al passo i+1 è dunque: Ri+1= Ri -qiV2^(-i).

Mantenendo questo procedimento avremmo dunque bisogno di uno shift register con shift variabile, in modo da poter effettuare uno shift diverso ad ogni passo.

Si è tuttavia ricavato un algoritmo alternativo, al fine di effettuare uno shift di una sola posizione ad ogni iterazione: invece di effettuare uno shift del resto parziale i+1 al termine delle operazioni, si effettua (ad ogni iterazione) uno shift a sinistra del dividendo parziale Ri e, con tale valore shiftato (che in binario coincide a 2Ri), si calcola Ri+1.

L’algoritmo alternativo è del tutto equivalente a quello standard, con l’unica differenza che richiede uno shift di un’unica posizione ad ogni iterazione, a differenza di quello standard, dove lo shift variava in base al passo i.

Valutiamo ora il numero di registri necessari per effettuare completamente le operazioni: il registro A, in cui immettiamo il dividendo, ad ogni iterazione viene svuotato di 1 bit (a causa degli shift verso destra dei dividendi parziali), mentre, il registro Q (contenente il quoziente) si riempie di 1 bit. Si può utilizzare quindi un unico registro AQ, il quale, al termine delle operazioni conterrà il resto negli N bit più significativi e nei restanti N-M+1 bit il quoziente.

Utilizzando tale algoritmo però, il registro AQ dovrà presentare la prima cella vuota, in modo da effettuare lo shift anche al primo passo senza perdere alcun bit.

Come visto dalla procedura manuale, l’operazione di divisione richiede una successione di sottrazioni e confronti; l’operazione di comparazione, idealmente, potrebbe essere effettuata mediante un componente comparatore. Tuttavia, anziché definire un nuovo componente, l’operazione di comparazione viene effettuata mediante un sottrattore, valutando il bit più significativo del risultato. Supponiamo infatti di voler effettuare un confronto tra A e B: effettuando l’operazione A-B, se il risultato presenta il bit più significativo, che chiameremo S per comodità, pari ad 1 (ovvero il risultato è negativo in una rappresentazione in complemento a 2), allora A<B, altrimenti A>=B.

Utilizzando questo metodo, possiamo sfruttare il sottrattore per effettuare anche la comparazione, il che porta un ulteriore vantaggio: siccome la comparazione avviene tra 2Ri e V, il risultato ci fornisce anche il valore di Ri+1, nel caso in cui 2Ri > V. Nel caso in cui, invece, 2Ri < V va effettuata un’operazione di ripristino del valore di Ri+1, riportandolo pari a Ri.

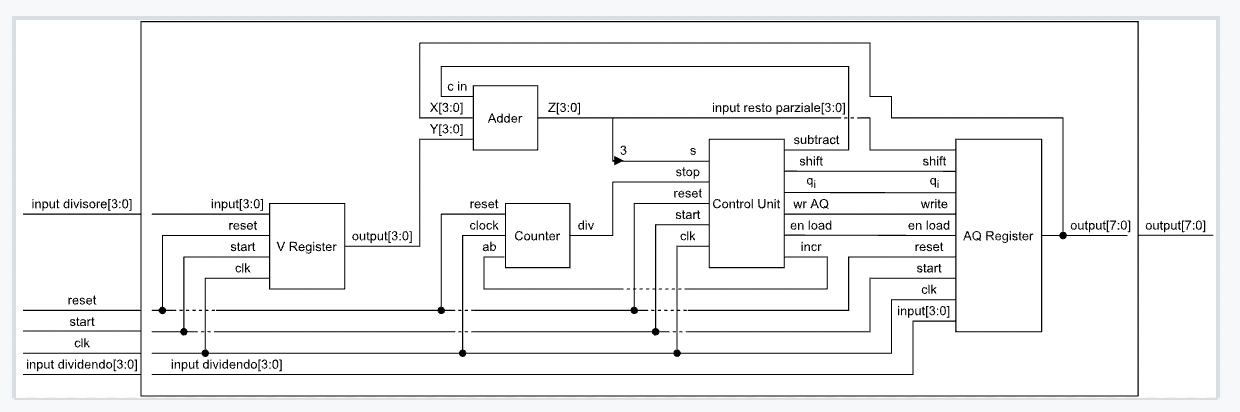
Per gestire tale rispristino ci sono 2 algoritmi possibili:

* Restoring: effettua il ripristino sempre al passo i-esimo; se il risultato della sottrazione da esito negativo, nello stesso passo si somma nuovamente V al valore di Ri+1, in modo da ottenere nuovamente il valore di Ri.
* Not restoring: effettua una correzione al passo i+1-esimo; tramite manipolazioni algebriche si è visto che, se al passo i+1-esimo effettuo una somma al posto di una sottrazione, ottengo lo stesso risultato del restoring.

Dato che entrambe le tecniche prevedono sia l’addizione che la sottrazione, avremo bisogno di un componente che implementa entrambe operazioni.

È bene notare che, a differenza dei moltiplicatori, i divisori sono implementati unicamente con un approccio sequenziale, dato che il risultato al passo i-esimo dipende dal risultato del passo i-1, non permettendo quindi un’implementazione di tipo parallelo.

Come tutte le macchine complesse, il divisore restoring è composto da una parte operativa ed una di controllo.



Parte operativa:

La parte operativa è stata progettata tramite un approccio strutturale, ed è composta da:

* Registro AQ: registro che inizialmente contiene il dividendo e che, al termine delle operazioni, conterrà il quoziente ed il resto.
* Registro V: registro che contiene il divisore.
* Sommatore RippleCarry: sommatore/sottrattore.
* Contatore: necessario ad indicare il termine delle operazioni da svolgere.

Componente RegistroAQ:

Immagine che contiene testo

Descrizione generata automaticamente

è stato progettato tramite un approccio comportamentale, dato che ci sono più segnali di controllo da gestire.

Implicitamente, con tale implementazione, è stato definita anche una priorità tra i segnali in ingresso, anche se ha poca rilevanza dato che, tali segnali, non saranno mai alti contemporaneamente (fa eccezione il solo segnale di reset, il quale potrebbe alzarsi in qualsiasi momento poiché proveniente dall’esterno e non dall’unità di controllo).

La dimensione ti tale registro è apri a 2N, dato che sia il resto che il quoziente saranno codificati su N bit.

Immagine che contiene testo

Descrizione generata automaticamente

Componente Registro V:

Immagine che contiene testo

Descrizione generata automaticamente

così come il registroAQ, anche tale registro è stato implementato mediante un approccio di tipo comportamentale.

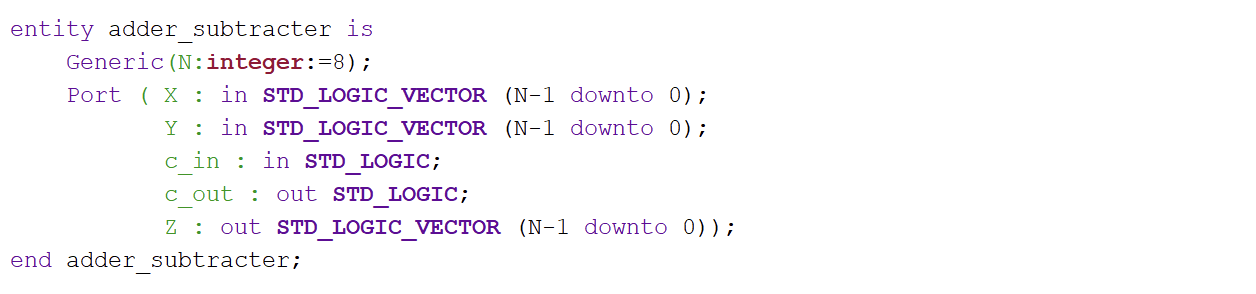
Immagine che contiene testo

Descrizione generata automaticamente

Componente Contatore:

tale componente è analogo a quello presentato nell’esercizio 9; pertanto si è ritenuto superfluo riportare nuovamente la sua implementazione.

Componente Adder-Subtracter:

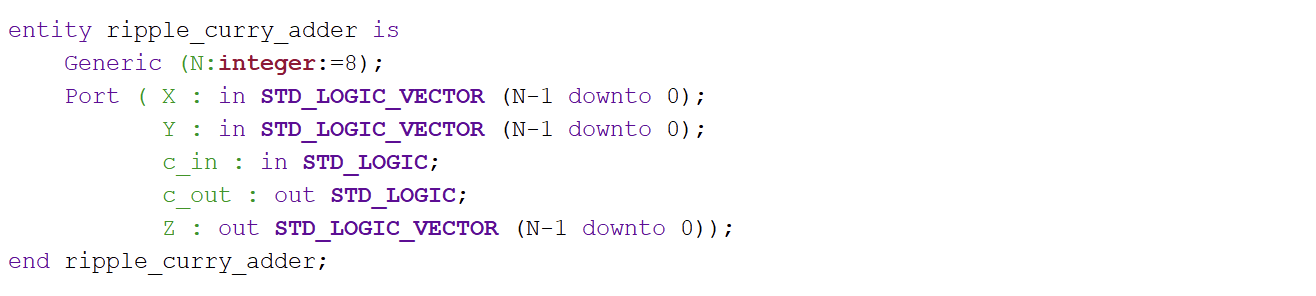


le operazioni necessarie per la divisione richiedono sia operazioni di somma che di differenza; pertanto, la scelta è ricaduta su un unico componente in grado di effettuare entrambe le operazioni. È stato progettato tramite un approccio strutturale, utilizzando un Ripple carry adder ed un signal che effettua il complemento a 2 di uno degli ingressi.

Immagine che contiene testo

Descrizione generata automaticamente

Component Ripple Carry adder:



Nonostante dal punto di vista prestazionale tale componente non sia tra i più efficienti, la scelta è ricaduta comunque su di esso data la sua struttura semplice e ritardi comunque trascurabili per operandi a 4 bit. È stato realizzato mediante interconnessione di componenti full-adder:

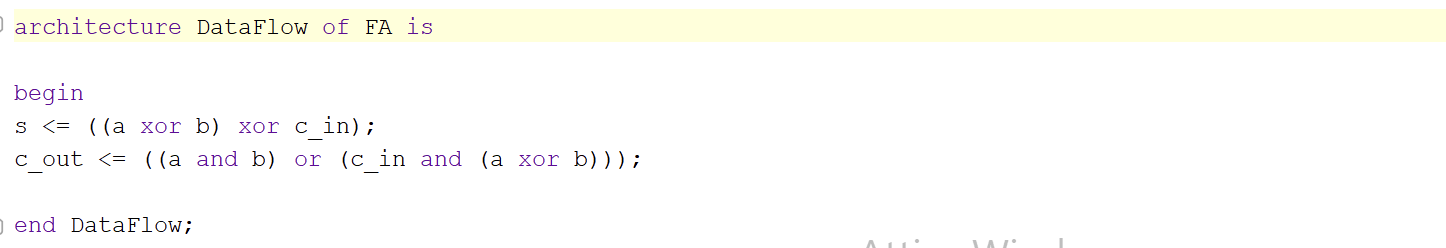
Immagine che contiene testo

Descrizione generata automaticamente

Componente full-adder:



è il componente base che ci permette di effettuare la somma su 2 bit; è stato progettato mediante approccio Dataflow:



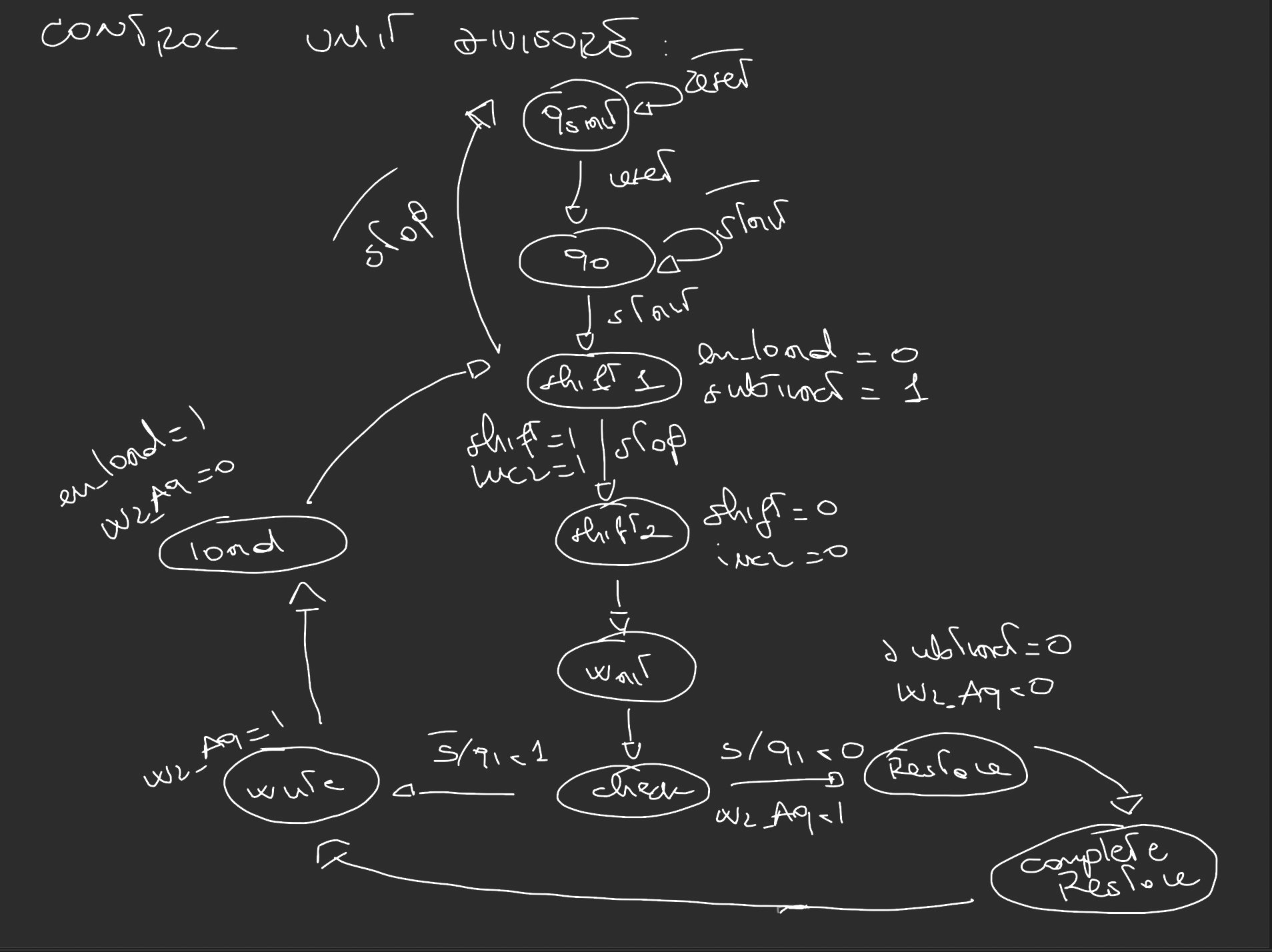
Al termine di tale panoramica sui componenti presenti nell’unità operativa, passiamo ora allo studio dell’unità di controllo.

Unità di controllo:

Immagine che contiene testo

Descrizione generata automaticamente

L’unità di controllo è stata realizzata come una macchina a stati:



Q\_start -> in tale stato si attende il segnale di reset. Quando arriva, si passa in q0.

Q0 -> in tale stato si attende il segnale di start. Quando questo arriva si passa nello stato shift.

Shift1-> a tale stato sono associate le uscite en\_load =0 e subtract=1. Queste servono a modificare le uscite dello stato di load: en load perché deve essere alto 1 colpo di clock, subtract perché di base la prima operazione che facciamo fare all’adder è la sottrazione per il confronto, poi se necessario viene effettuata l’addizione per il restore.

Si effettua poi un controllo sul valore di stop, corrispondente all’uscita div del contatore: se questa è alta, ovvero abbiamo effettuato tutte le operazioni, non si effettua un nuovo shift ma si ritorna nello stato di start e ci si pone in attesa di un reset. Se invece il valore del div è basso, si passa allo stato di shift2 e si alzano i segnali di shift, per abilitare lo shift del registro AQ, e si alza il segnale di abilitazione per il conteggio del contatore.

Shift2-> in tale stato semplicemente si abbassano i segnali alzati nella transizione, poiché vogliamo siano alti solo 1 colpo di clock per evitare conteggi o shift spuri. A questo punto si passa allo stato di wait.

Wait -> in tale stato si attende semplicemente che il sommatore abbia effettuato il confronto. Siccome è una macchina puramente combinatoria e non ha nessun segnale di abilitazione, il sommatore darà sempre un valore in uscita; sta a noi aspettare il tempo necessario, per far si che il risultato sia relativo agli operandi immessi e non ai precedenti. Il sommatore ha il suo ritardo dovuto alla propagazione, per questo è necessario tale stato. Con operandi a 4 bit, questo stato intermedio è sufficiente a garantire che il risultato in uscita sia quello corretto; tuttavia, all’aumentare del numero di bit, potrebbe essere necessario aggiungere un nuovo stato oppure abbassare la frequenza del clock. Si passa poi allo stato di check.

Check -> in tale stato si effettua il controllo sul confronto avvenuto tramite sottrazione:

* se il bit più significativo è basso, ovvero il risultato è positivo, si passa allo stato di write e non è necessario il restore. Si pone, inoltre, il valore del quoziente parziale pari a 1.
* se il bit più significativo è basso, ovvero il risultato è negativo, si passa allo stato di restore.

Si pone, inoltre, il valore del quoziente parziale pari a 0 e si alza il segnale di write\_aq, al fine di scrivere il valore in uscita dal sommatore, necessario per il restore.

Restore -> in tale stato si abbassa il valore di subtract, poiché è necessario fare la somma per effettuare il restore, e si abbassa inoltre il segnale di wr\_aq, per evitare sovrascritture indesiderate. Si passa poi allo stato di complete restore.

Complete Restore-> tale stato ha sostanzialmente lo stesso scopo dello stato di wait, ovvero dare il tempo all’adder di effettuare il restore e prendere il risultato Z corretto. Si passa poi allo stato di write.

Write -> tale stato scrive l’uscita Z dell’adder nella parte del registro AQ dedicata ad A, che ovviamente sarà diversa in base a se viene effettuato il restore o meno: nel primo caso il valore scritto coinciderà con il precedente, altrimenti sarà pari al valore della differenza calcolata. Si alza quindi il segnale di wr\_aq e si passa nello stato di load.

Load -> in tale stato si abbassa il segnale di wr\_aq e si alza quello di en\_load, per caricare nella parte dedicata a Q il bit del risultato parziale. Si ritorna poi nello stato di shift1.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

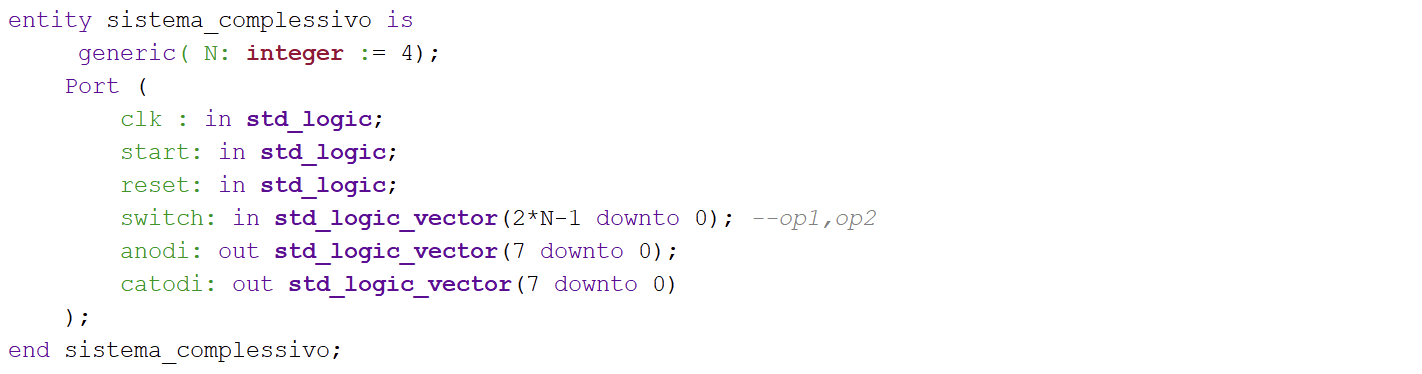
Immagine che contiene testo

Descrizione generata automaticamente

Implementazione su fpga:

Per l’implementazione su scheda sono stati introdotti 2 debouncer per i segnali di start e reset ed il componente display, necessario per la visualizzazione dei risultati.

Il problema principale del passaggio da simulazione ad fpga, per un progetto di questo tipo, riguarda la frequenza del clock: quest’ultimo potrebbe essere troppo veloce e non dare il tempo all’adder di svolgere le operazioni e di fornire il risultato corretto. Nel nostro caso, dato che gli operandi sono su soli 4 bit, non è stato necessario introdurre un divisore di frequenza; all’aumentare però della dimensione degli operandi, potrebbe tuttavia essere necessario introdurlo.

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente

Immagine che contiene testo

Descrizione generata automaticamente